대규모 분산 처리 프레임워크에 따른 대규모 그래프 처리 성능 비교

배경숙, 공용준, 심탁길, 신의섭, 성기진
SKC&C, 인프라 사업개발 본부, 클라우드 컴퓨팅 개발담당 팀
e-mail : pavin@sk.com, andrew.kong@sk.com, terryshim@sk.com
shine62@sk.com, keejin.seong@sk.com

A Performance Comparison of Distributed Data Processing Frameworks for Large Scale Graph Data

Kyung-sook Bae, Yong-joon Kong, Tak-kil Shim, Eui-seob Shin, Kee-kin Seong
Cloud Computing Team, Dept. of Infra Business Dev, SKC&C

요 약

최근 IT 분야의 화두로 '빅 데이터' 가 떠오르고 있으며 많은 기업들이 이를 분석하여 이익을 증대하기 위한 노력을 하고 있다. 이에 구글은 초기에 맵리듀스라고 하는 대용량 분산처리 프레임워크 기술을 확보하여 이를 기반으로 한 서비스를 제공하고 있다. 그러나 스마트 단말 및 소셜미디어 등의 출현으로 다양한 디지털 정보들이 그래프로 표현되는 추세가 강화되고 있으며 기존의 맵리듀스로 이를 처리하는 데에 한계를 느낀 구글은 Pregel 이라는 그래프 형 자료구조에 최적화된 또 다른 분산 프레임워크를 개발하였다. 본 논문에서는 일반적인 그래프 형 데이터가 갖는 특성을 분석하고, 대용량 그래프 데이터를 처리하는데 있어 맵리듀스가 갖는 한계와 Pregel 은 어떤 방식으로 이를 극복하고 있는지를 소개한다. 또한 실험을 통하여 데이터의 특성에 따른 적절한 프레임워크의 선택이 대용량 데이터를 처리하는 데에 있어서 얼마나 큰 영향을 미치는지 확인한다.

1. 서론

최근 구글과 야후 등 인터넷 기업과 학계 전문가들은 최근 미국 샌디에고에서 개최된 지식 개발과 마이닝(knowledge discovery and Data Mining 2011) 컨퍼런스에서 폭발적으로 증가하는 데이터와 다양한 형태로 생산되는 정보의 분석 방향에 대해 논의 하였다.[2] 오늘날 기업들은 인터넷의 폭발적 성장으로 이용자와 이용자의 행동에 대한 막대한 양의 데이터를 수집하고 있으며, 이를 분석하여 기업의 이익을 증대하기 위한 노력을 하고 있다.[2][3] 이러한 측면에서 구글은 초기에 맵리듀스(MapReduce)라고 하는 대용량 분산처리 프레임워크 기술을 확보하여 이를 기반으로 한 서비스를 제공하였다. 이후, 맵리듀스의 오픈 소스 구현체인 하둡(Hadoop)이 개발되었고 실제로 이를 구축하고 서비스를 제공하는 사례가 많이 생겨나고 있다.[3][4]

오늘날의 데이터들은 이전의 정형화된 데이터베이스와는 달리 대부분 네트워크에 산재된 형태로 존재하며, 특히 소셜미디어의 출현으로 인해 디지털 정보들이 그래프들로 표현되는 추세가 강화되고 있다. 뿐만 아니라, 전력 파워 그리드, 수자원 분배 시스템, 교통관리 시스템 등과 같은 복잡한 엔지니어링 시스템에서 발생되는 정보 또한 그래프로 표현된다.[2][3] 이러한 그래프 문제들을 해결하기 위해 맵리듀스를 효과적으로 활용하기 위한 연구가 활발히 진행되었으나, 임시 결과를 저장하고 이를 원격 읽기(remote read) 해야 하는 맵리듀스의 특성상 같은 작업을 반복 처리해야 하는 그래프 문제를 처리하기에는 속도 면에서나 프로그래밍 복잡도 면에서 부적합하다.[7] 이러한 이유로 구글은 대규모 그래프 데이터를 처리하기 위한 프레임워크인 Pregel 을 개발하고 이를 활용한 서비스를 제공하고 있다.

본 논문에서는 분산 처리 프레임워크의 종류에 따른 그래프 알고리즘의 성능을 비교한다. 비교에 사용한 프레임워크는 앞서 언급된 맵리듀스와 Pregel 이 사용되었으며 성능을 측정하기 위한 그래프 알고리즘으로는 구글의 페이지랭크 알고리즘을 사용하였다. 논문은 총 5 절로 구성되어 있다. 2 장에서는 구글의 대규모 데이터 처리 프레임워크인 맵리듀스와 그래프 기반의 프레임워크인 Pregel 에 대해 소개하고 3 장에서는 웹 페이지의 중요도를 측정하기 위한 알고리즘인 페이지랭크에 대해 소개한다. 4 장에서는 Pregel 기반의 페이지랭크와 맵리듀스 기반의 페이지랭크의 성능을 비교, 분석하고 5 장에서 결론으로 마무리 한다.

2. 맵리듀스와 Pregel

2 장에서는 대용량 분산처리 프레임워크인 맵리듀스와 Pregel 에 소개하고 각각의 특징에 대해 설명한다.

2.1 맵리듀스

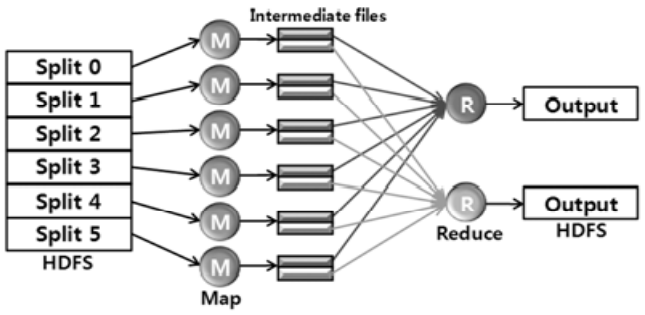
맵리듀스는 구글에서 개발한 대규모 데이터를 병렬처리하기 위한 소프트웨어 프레임워크로서 많은 저가의 장비로 페타바이트 이상의 데이터를 처리하는 일종의 프로그래밍 모델이다.[1][8][9]

map (k1, v2) → list (k2, v2)
reduce (k2, list (v2)) → list (v2)

(그림 1) 맵리듀스 개념

먼저 그림 1 의 맵리듀스 프로그램 모델에 대해 간단히 살펴보자. 맵 함수는 입력으로 키(k1)와 값(v1)을 전달받아 처리한 후, 출력으로 새로운 키(k2)와 값(v2)의 목록을 출력한다. 맵 함수가 반복적으로 수행되면 여러 개의 출력 데이터가 생성되는데, 이 출력 데이터를 키로 정렬하면 각 키에 여러 개의 데이터가 존재한다. 이 키(k2)와 값 목록(list(v2))이 리듀스 함수의 입력이 되고, 리듀스 함수는 이를 처리하여 값을 출력한다.

맵리듀스는 마스터/슬레이브 구조를 가지는데 마스터는 태스크를 할당하고 제어하는 역할(Jobtracker)을 하고, 슬레이브는 실제 태스크를 수행하는 역할(Tasktracker)을 한다. 슬레이브가 수행하는 태스크가 바로 맵(map)과 리듀스(reduce) 태스크이다.



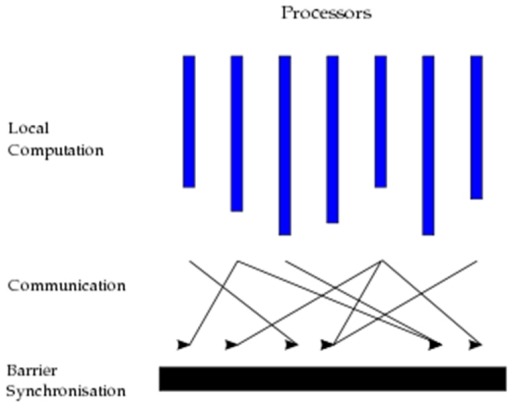
(그림 2) 맵리듀스 처리 흐름

그림 2 는 맵리듀스의 동작을 순차적으로 설명하고 있으며 맵과 리듀스 태스크간의 관계를 보여준다. 파일은 스플릿(Split)이라고 하는 블록으로 나누어져 저장되어 있고 각 Tasktracker 는 블록을 읽어 맵 태스크를 수행한다. 맵의 결과는 중간 파일 형태로 각각의 노드에 저장되며, 리듀스 태스크를 수행하는 노드는 다수의 노드로부터 맵의 결과를 전달받아 최종 결과를 생성한다. [5]

여기에서 주목해야 할 점은 맵 프로세스가 생성한 중간 결과를 로컬 파일 시스템에 저장한다는 점이다. 일반적으로 그래프 알고리즘은 vertex 간에 연결된 에지를 이용하여 정보를 전달하고 전달받은 정보를 이용하여 상태를 갱신하는 일련의 과정을 반복하여 문제를 해결한다. 맵리듀스 기반으로 그래프 알고리즘을 구현하면, 맵 프로세스는 정보교환을 위해 매 iteration 마다 로컬쓰기와 원격읽기를 반복해야만 하므로 성능의 지연이 발생된다.

2.2 Pregel

그래프는 버텍스(vertex)와 방향을 가진 에지(edge)로 표현되는 데이터로서, 일반적으로 그래프 문제들은 에지와 버텍스의 연결관계를 이용하여 메시지를 주고받으며 비슷한 패턴의 반복연산을 수행하며 문제를 해결해 나간다. 그렇기 때문에 메모리 로컬리티(locality)를 보장하는 것이 그래프 문제의 성능을 결정하는 중요한 요소이며, 대용량 그래프 분석을 위한 분산 환경에서는 이를 만족시키기 어렵다. 예를 들어, 기존의 분산 컴퓨팅 플랫폼인 맵리듀스의 경우에 대규모의 순차적 데이터의 처리에는 적합하지만 분산된 노드간에 메시지 교환을 위해서 로컬 쓰기와 원격 쓰기를 반복해야 하므로 성능이 저하된다. 이와 같은 문제를 해결하기 위해 구글은 그래프 분석 전용 프레임워크를 개발하였는데 이것이 바로 Pregel 이다.



(그림 3) Bulk Synchronous Parallel

Pregel 은 그림 3 의 BSP(Bulk Synchronous Parallel Algorithm) 이라는 병렬처리 기법에서 아이디어를 얻어서 개발되었다. 그래프 문제에서 흔히 사용되는 iteration 을 Pregel 에서는 superstep 이라는 용어로 정의한다. BSP 모델에서는 매 iteration 마다 barrier 를 두어 프로세스간에 동기화를 한다. Pregel 의 barrier 는 모든 버텍스가 완료해야 하는 작업을 의미하며 superstep 의 시작을 알리거나, 메시지의 교환완료, 버텍스의 계산완료, superstep 의 종료 등의 상태를 공유하여 분산되어 있는 버텍스들을 동기화 한다. [7]

Pregel 는 메시지 교환 방식에 있어서 맵리듀스와 크게 차별화 된다. 맵리듀스가 버텍스간에 메시지 교환을 위해 파일의 원격 읽기 방식을 사용한 반면, Pregel 은 메시지 패싱 모델을 사용하여 가상으로 메모리를 공유하여 메시지를 교환한다. 대용량 데이터

를 처리하는 분산 환경임을 고려할 때, 메시지 교환 방식이 맵리듀스와 Pregel 의 성능 차이를 좌우한다.

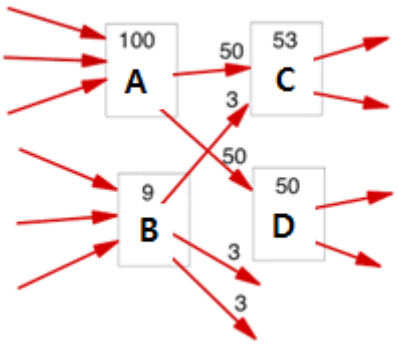
Pregel 역시 맵리듀스와 마찬가지로 마스터/슬레이브 구조를 가지며, Fault Recovery 기능을 가지고 있다. 모든 분산 노드들은 마스터의 명령에 의해 모든 Superstep 이 종료될 때마다 자신의 상태(버텍스의 값, 에지 정보, 메시지 정보 등)를 분산파일 시스템에 저장하고, 마스터는 모든 슬레이브의 작업을 모니터링 한다. Fault 가 발생한 경우, 마스터는 슬레이브에게 마지막 성공한 superstep 으로 돌아가서 작업을 이어받아 진행하도록 명령하여 Fault 에 대처한다.[7]

Pregel 의 오픈 소스 구현체로 ravel 사의 GoldenOrb 와 아파치의 Giraph 등이 있으며, 모두 파일 시스템으로는 HDFS 를 사용하며 동기화를 위해서는 아파치의 zookeeper 를 사용한다.

3. 페이지랭크

웹 페이지의 중요성은 본질적으로 주관적인 문제여서 읽는 사람의 관심사나 지식 그리고 태도 등에 의존한다. 구글의 페이지랭크는 수많은 웹 페이지의 중요도를 객관적으로 평가하기 위한 한 방법으로서 웹 페이지의 콘텐츠와는 상관없이 오직 웹의 링크 구조만을 사용하는 전형적인 그래프 알고리즘이다.[6]

페이지랭크는 일반적으로 링크가 많이 된 페이지 일수록 그렇지 못한 페이지보다 더 중요하며, 또 중요한 페이지로부터 링크를 받는 페이지 일수록 더 중요한 페이지라는 관찰에 기초한다. 즉, 단순히 인용 횟수만으로 페이지의 중요도를 파악하는 방식 이상의 더 정교화된 방법을 제시한다.[6]



(그림 4) 페이지랭크 계산

그림 4 에서 페이지 C 의 랭크 값을 계산하는 과정을 간단히 살펴보자. 페이지 C 는 페이지 A 와 B 로부터 링크를 받고 있고 A 와 B 는 각각 2 개와 3 개의 포워드 링크를 가지고 있다. 페이지 A 와 B 는 자신이 가리키는 페이지들의 랭크에 균일하게 기여하기 위해, 자신의 랭크 값을 포워드 랭크 개수로 나누어 준다. 따라서 C 는 A 랭크 값의 1/2 을, B 랭크 값의 1/3 을 받아 53 이라는 랭크 값을 갖게 된다.

어떤 웹 페이지 A 를 가리키는 페이지들의 집합을 $V=\{v_1, v_2, \ldots, v_n\}$ 라 하고 어떤 웹 페이지 X 로부터 나가는 링크의 개수를 N(X)라고 하면 페이지 A 의 단순 랭크 값은 아래 수식으로 정리될 수 있다.

$$PR(A) = c \sum_{i=0}^{n} \frac{PR(Vi)}{N(Vi)} \quad \text{식(1)}$$

c 는 전체 웹 페이지의 랭크 총합을 일정하게 하기 위해 사용되는 정규화 팩터이며 c<1 이어야 한다. 이는 포워드 링크가 없는 페이지를 가리키는 페이지(댕글링 링크)의 가중치가 어디로 분산되고 있는지 불분명하기 때문에 그런 페이지들을 시스템 속에서 사라질 수 있도록 하기 위함이다.[] 그 밖에, '랭크싱크'로부터 초래되는 문제를 해결하기 위해서나 랜덤 서퍼 모델에 기반하는 댐핑 팩터 등을 반영하여 여러 가지 변형의 수식을 도출할 수 있다.[6]

모든 페이지에 초기 값을 주고 위의 수식을 이용하여 수렴할 때까지 반복적으로 연산함으로써 최종 결과를 얻을 수 있다. 초기값을 어떻게 주느냐는 수렴의 속도에는 영향을 주지만 수렴할 때까지 반복작업을 계속한다면 최종 값에는 영향을 끼치지 않는다. 보통 수렴할 때까지의 반복회수는 50 회 내외로 알려져 있다 [6]

4. 실험 및 결과

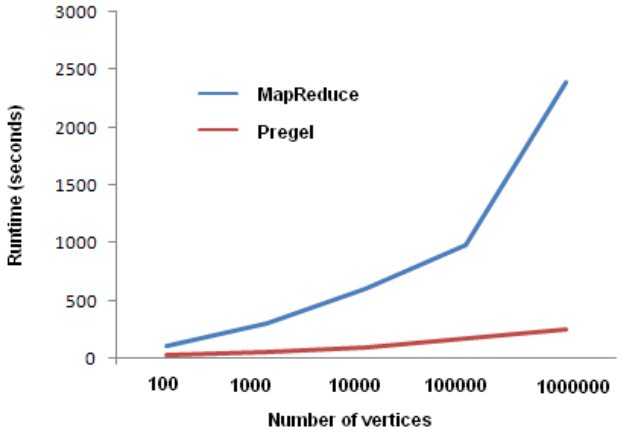
본 논문에서는 대규모 그래프 데이터의 처리에 있어 소프트웨어 프레임워크의 종류에 따른 성능을 비교하기 위해 맵리듀스 기반의 그래프 분석 알고리즘과 Pregel 기반의 그래프 분석 알고리즘의 처리속도를 측정하였다.

4.1 실험환경

실험에 사용된 시스템 환경은 다음과 같다. Intel QuadCore 로 구성된 3 대의 서버를 사용하였으며, 운영체제로는 2.6.18 커널 버전의 리눅스를 설치하였다. 대용량 데이터 저장과 처리를 위한 파일 시스템으로는 하둡의 분산 파일 시스템(HDFS)를 사용하였다. 실험에 사용한 맵리듀스 프레임워크로는 아파치의 하둡, Pregel 프레임워크로는 아파치의 Giraph 를 사용하였으며 그래프 분석 알고리즘으로는 3 장에서 소개한 페이지랭크 알고리즘을 사용하였다. Pregel 의 동기화를 위해 Zookeeper 를 한 대의 노드에 설치하여 사용하였다. 실험을 위해 맵리듀스의 Mapper 의 수와 이에 대응되는 Pregel 의 Worker 의 수를 같게 설정하고 vertex 의 수를 증가시키며 동일한 그래프에 대한 처리속도를 측정하였다.

4.2 실험 결과 및 분석

입력 그래프는 vertex 의 수를 100 개에서 1 백만 개까지 증가시키며 처리속도를 측정하였으며, 반복횟수는 50 으로 고정하여 테스트 하였다. 실험결과, 아래 그림 5 와 같이 Pregel 기반의 페이지랭크가 맵리듀스 기반의 페이지랭크보다 월등히 높은 처리속도를 보였으며, 그래프의 크기가 클수록 그 차이는 더욱 커짐을 알 수 있었다.



(그림 5) 그래프 크기에 따른 처리속도 비교

5. 결론

본 논문에서는 분산 처리 프레임워크에 따른 그래프 알고리즘의 처리 성능을 비교하였다. 같은 작업을 반복 처리하고 각 단계마다 빈번한 메시지 교환을 필요로 하는 그래프 문제의 특성상, 파일 읽기/쓰기로 정보를 교환하는 맵리듀스 보다 메시지 패싱 방식을 사용하는 Pregel 이 더 좋은 성능을 보이는 것을 확인할 수 있었다. 현재까지 대규모 데이터의 처리에 있어 맵리듀스 프레임워크가 독보적인 위치를 차지하였으나 앞으로는 데이터의 특성을 고려하여 프레임워크를 선택하는 것이 분석 알고리즘의 경쟁력을 결정짓는 중요한 요소가 될 것이다.

참고문헌

[1] 하둡 기술 연계한 데이터 분석, 김희배, 2011.9
[2] Big Data Analytics, Gartner, 2011 년 1 월
[3] Big Data : The Next Frontier for Innovation, Competition, and Productivity, McKinsey & Company, 2011 년 5 월
[4] Managing Big Data with Hadoop & Vertica, Vertica Systems, 2009 년 10 월
[5] 유대현, 정상화, 김태훈, "Hadoop 기반 분산 컴퓨팅 환경에서 네트워크 I/O 의 성능개선을 위한 TIPC 의 적용과 분석", 정보과학회논문지, 제 36 권 제 5 호, pp.351~359, 2009.10
[6] 이명헌 경영스쿨 http://www.emh.co.kr/xhtml/google_pagerank_citation_ranking.html
[7] Grzegorz Malewicz, Matthew H Austern, "Pregel: A System for Large-Scale Graph Processing", Proceedings of the 28th ACM symposium on Principles of distributed computing, 2009
[8] Hadoop MapReduce Tutorial, Available: http://hadoop.apache.org/core/docs/r0.18.0/mapred_tutorial.Html
[9] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," COMMUNICATIONS OF THE ACM, vol.51, no.1, pp.107-113, January 2008.
[10] Hadoop Wordcount Example, Available: http://wiki.apache.org/hadoop/WordCount
[11]Hadoop Cluster Setup Document, Available: http://hadoop.apache.org/core/docs/r0.18.0/cluster_setup.html